ITOS Configuration Monitor Users' Guide

Integrated Test & Operations System
5 October 2006

1 Overview

The configuration monitor allows you to do two things: First, you can assign to a telemetry mnemonic the value of a STOL-language expression. Second, you can execute any STOL directive if a STOL-language expression evaluates to a non-zero ("true") value. In this document, we'll call the former an *equation*, and the latter a *condition*.

"What's it for?" you ask. Well...

Suppose you're telemetering current and voltage from the spacecraft, but need to use power on displays and in STOL procs. You can define a mnemonic for power and use an equation to calculate it.

Suppose you wanted to capture some data when a certain on-board device was on. You could use a condition to detect when the device comes on, and to start a sequential print to a file. Another condition could detect when the device goes off, and stop the print.

Finally, suppose you wanted to represent the health of an on-board subsystem with a single, good/bad indicator; or display one of several states a subsystem may be in based on several variables. A combination of equations and conditions might be just the thing.

We consider this a preliminary release of this new capability, and we are open to suggestions from you regarding how we might improve on this initial offering. We have several ideas of our own, which we'll ask you about directly at appropriate points in this document.

Let's jump right into an example. Suppose I put the following two lines in a text file called '`buspwr.cfg`':

```
pbuspwr = (pbuscurr * pbusvolts)
if (pbuspwr .gt. 35.2) then start someproc
```

Then I execute the STOL directive `cfgmon buspwr` to start the configuration monitor processing that file.

The configuration monitor arranges to get a copy of the value of the mnemonics `pbuscurr`, `pbusvolts`, and `pbuspwr` whenever each is assigned a different value from its previous value. When it gets new a value for `pbuscurr` and `pbusvolts`, it multiplies them together and assigns the results to the mnemonic `pbuspwr`. The new `pbuspwr` value is limit-checked just as if it had been received from the spacecraft.

When `pbuspwr` is updated, the new value is compared against 35.2, according to the condition on the second line. If the mnemonic's value is greater, the configuration monitor sends the directive `start someproc` to the STOL interpreter to start the STOL proc called '`someproc`'.

2 Equations

Configuration monitor equations have the form:

mnemonic = STOL *expression*

where the *mnemonic* is a telemetry mnemonic, and the STOL *expression* is a mathematical or logical STOL-language expression containing at least one telemetry mnemonic.

The newline character at the end of the STOL *expression* ends the equation. Newlines contained in a parenthesized expression are ignored, so we recommend that you always parenthesize the right-hand side of the equation. Newlines on either side of the equals sign also are ignored.

So, for example, the following equations will be interpreted correctly:

```
mymnemonic = hismnemonic * hermnemonic + tan(theirmnemonic)
mymnemonic = (hismnemonic * hermnemonic + tan(theirmnemonic))
mymnemonic
        = hismnemonic * hermnemonic + tan(theirmnemonic)
mymnemonic = (hismnemonic
                 * hermnemonic + tan(theirmnemonic))
mymnemonic = hismnemonic * hermnemonic + \
                tan(theirmnemonic)
mymnemonic =
        (hismnemonic * hermnemonic + tan(theirmnemonic))
```

whereas this next set of equations will **not** parse correctly:

```
mymnemonic = hismnemonic * hermnemonic +
                tan(theirmnemonic)
```

3 Equations in Decom

ITOS users can create a configuration file that lists telemetry mnemonics whose values are the result of equations. These equations may contain other mnemonics within the equation. These equations are evaluated at the exact moment the packets are decommed.

The configuration file identified by `GBL_PSEUDO_TLM_FILE` contains the definitions of the equations and the mnemonics they are applied to. If this global mnemonic contains just the file name then the file must reside in the directory that you start ITOS in. Otherwise set this global to the explicit path of the file.

The format for each line in the configuration file is:

```
appid mnemonic = expression

Where:

appid = Any valid appid defined in the database. Mnemonics in the equation should
        occur in this packet since the equation will be evaluated when this packet
        is decommed.
mnemonic = Any mnemonic defined in the database that is NOT part of a packet. Should b
           a pseudo mnemonic created specifically for this assignment.
equation = Any valid STOL expression that can contain any mnemonic defined in the data
```

Example of configuration file follows:

```
1 my_mnemonic_1 = .0835 + (pkt1_mnem1 - pkt1_mnem2) * 2047
1 my_mnemonic_2 = 2.345 + 2045/(pk1_mnem3 - my_mnemonic_1) * 3.5
3 my_mnemonic_3 = 33.34 + (pkt3_mnem1 - pkt3_mnem2) * 2047
3 my_mnemonic_4 = 2.345 + 2045/(pk3_mnem3 - my_mnemonic_3) * 3.5
```

Notes: At the start of acquiring telemetry there will be a message from the tlmClient application noting the configuration file that was found. If no file is found there will be a warning message but telemetry decom will continue.

4 Conditions

Configuration monitor conditions have the form:

`if` STOL *expression* `then` *action*

where the STOL *expression* is a logical or mathematical STOL-language expression and *action* is a STOL directive or the configuration monitor's special `alert` action, discussed below.

If the *action* begins with the word `alert`, the rest of the statement is sent out as an event message, with event type CFG_ALERT.

The newline character at the end of the *action* ends the condition statment. Newline characters in the action may be escaped with a backslash (\), and they are ignored inside a quoted string. Newlines anywhere between the `if` and the beginning of the action are ignored.

For example, the following conditions will be parsed correctly:

```
if (mymnemonic .gt. yourmnemonic) then alert me
if mymnemonic .gt. yourmnemonic then alert me
if
mymnemonic
.gt.
yourmnemonic
then
alert me
if (mymnemonic .gt. yourmnemonic) then alert \
        me
```

but this final example will not parse correctly:

```
if mymnemonic .gt. yourmnemonic) then alert
    me
```

We anticipate you will desire the ability to print formatted alert messages containing mnemonic values. One way we plan to implement this is by adding a new STOL directive and/or function which will issue event messages and using the existing STOL `format` function to format it. Alternatively, or in addition, we might implement a page-definition-style format string for displaying data.

We also anticipate the need to handle an *action* which consists only of an equation. While it is acceptable to simply pass the equation to the STOL interpreter like any other action, it might be preferrable to handle the equation right in the configuration monitor.

5 Options and Comments

Comments in configuration monitor files begin with the pound sign ('#') or semicolon (';') and continue to the end of the line.

Each condition or equation may be preceded by one or more options enclosed in curly brackets, and separated by commas. As an example, here is all of the available options set to their default settings:

```
{ count = 1, tagtype = changed, trigger = norm, partial = fill }
```

These options control how the configuration monitor obtains and handles the data it needs to evaluate the STOL expressions.

Options are applied to all subsequent statements in a definition file. If you set some options at the beginning of the file and don't set them again in that file, the initial option settings apply to every line in the file.

For example, if a configuration file consists of:

```
{ count = 1 } if (gbl_tlmrate .gt. 2.25e6) then alert "high rate"
if (gbl_tm_frmrate .gt. gbl_tlmrate) then alert "nonsense rate"
{ count = 2 } if (gbl_tlmrate .eq. 0) then alert "telemetry stopped"
```

The `count` option is set to one applies to the first two entries, and it is set to two for the third entry. Any subsequent entry would have count set to two, unless it specified a different count, which then in turn would apply to all entries following it, etc.

The specifics of each options is dealt with in the following sections.

5.1 Count

The `count` option applies only to conditions. It gives the number of consecutive times the given expression must evaluate non-zero ("true"), before the associated action is taken. The default value is one. This option may be used to filter out spurious or transient results.

The configuration monitor keeps a count of the number of times consecutively that the expression is evaluates to a non-zero value ("true"). Whenever the expression evaluates to zero ("false"), the consecutive-true counter is reset to zero.

Note that the action is take only when the consecutive-true counter is equal to the value stored by the `count` option. This means that the action is taken only once, as long as the condition remains true. (Please let us know if you require an option such that the action is taken every time the expression evaluates to true, even if the last evaluation also gave a true result.)

Note also that this option may interact with the `tagtype` option. If the condition is not triggering on a timed interval, and if the `tagtype` is set to the value `changed` (the default), only new values different from the current value for each mnemonic are sent to the configuration monitor. So, as long as the right-hand-side values are unchanged, the expression will not be re-evaluated. In this state, if `count` is greater than 1, it may not be reached even though the expression would evaluate true many times if the `tagtype` were `set`.

5.2 Tagtype

The `tagtype` options controls when new values are sent to the configuration monitor when the `trigger` option is not `time`. The `tagtype` option may be set to `changed`, the default, or to `set`.

The `changed` tagtype indicates that the configuration monitor should be sent a mnemonic's value only when it differ from its current value. The `set` tagtype indicates that the configuration monitor should be sent a mnemonic's value whenever it is updated, usually by unpacking telemetry packets.

You also can use the tagtypes `flag_changed` and `flag_set` to receive values when the mnemonic flags change or are updated, respectively. Mnemonic flags indicate things such as limit violations, questionable quality, and static data. One of these must be given if you use the `isstatic` function in an expression, since the static flag is set when the data is not updating!

Multiple values are specified using the C-language OR operator (`|`), as in the following example.

```
tagtype = changed | flag_changed
```

This option interacts with the `count` option. See Section 5.1 [Count], page 5.

5.3 Trigger

The `trigger` option controls when the STOL expression is evaluated. Triggering may be data driven, or may occur on a timed interval.

To evaluate equation or condition expressions on a timed interval, set the `trigger` option to an integer value. This is the number of milliseconds between evaluations.

Data-driven evaluation is controlled by setting the `trigger` option to one or more of the following values:

`norm`	Evaluate the expression when values for all mnemonics in it have been updated since the last evaluation, or when a second update occurs on at least one mnemonic value.
`eom`	Evaluate the expression whenever we have received at least one updated value. The evaluation is done after a telemetry packet has been fully unpacked, so any expression mnemonics contained in the packet will have been updated.
`pktxxx`	Evaluate the expression whenever the telemetry packet with application ID `xxx` has been unpacked, provided at least one mnemonic value has been updated since the last expression evaluation. **This option has not yet been implemented.**

Multiple values are specified using the C-language OR operator (`|`), as in the following example.

```
trigger = norm | eom
```

We anticipate adding a mnemonic triggering option similar to the `pktxxx` triggering option. In this scenario, the expression would be evaluated whenever a value is received for a given mnemonic.

5.4 Partial

The `partial` option controls what the configuration monitor does when it goes to evaluate an expression and finds it has not received updated values for all of the mnemonics in the expression. If it is set to `fill`, missing values are read from the ITOS system's Current Value Table (CVT). If `partial` is set to `discard`, the expression will not be evaluated if any mnemonic values have not been updated.

Note that this option is meaningless if triggering is on a timed interval. In this mode, all values are read from the CVT.

6 Debugging

The configuration monitor definition parser doesn't always produce terribly helpful error messages. If you get stuck, you can embed a `debug` statement in the definition file. You'll probably still be stuck, because it takes a developer to interpret the debug messages, but at least you have a chance.

Just put the word "debug" on a line by itself (outside of any equation or configuration or option set), and you'll get a ton of inscrutable debugging messages. Each occurrence of the `debug` statement toggles debugging on or off.

For example, the configuration:

```
debug
psbatpwr = (psbatcurr * psbatvolt)
debug
if (psbatpwr .gt. 35.2) then start someproc
```

produces the following output, which consists mainly of debugging messages for the '`psbatpwr`' equation:

```
state stack now 0
Entering state 5
Reducing via rule 5 (line 133), debugcmd  -> entry
state stack now 0
Entering state 4
Reducing via rule 1 (line 127), entry  -> definition
state stack now 0
Entering state 3
Reading a token: --accepting rule at line 159 ("
")
--accepting rule at line 143 ("psbatpwr")
Next token is 259 (TOK_WORD)
Reducing via rule 17 (line 270),  -> options
state stack now 0 3
Entering state 8
Next token is 259 (TOK_WORD)
Shifting token 259 (TOK_WORD), Entering state 12
Reading a token: --accepting rule at line 69 (" ")
--accepting rule at line 133 ("=")
Next token is 61 ('=')
Shifting token 61 ('='), Entering state 24
Reading a token: --accepting rule at line 69 (" ")
--accepting rule at line 209 ("(")
--accepting rule at line 246 ("psbatcurr")
--accepting rule at line 69 (" ")
--accepting rule at line 246 ("*")
--accepting rule at line 69 (" ")
--accepting rule at line 246 ("psbatvolt")
--accepting rule at line 217 (")")
Next token is 261 (TOK_SX)
```

```
Shifting token 261 (TOK_SX), Entering state 32
Reducing via rule 7 (line 141), options TOK_WORD '=' TOK_SX  -> equation
state stack now 0 3
Entering state 6
Reducing via rule 3 (line 131), equation  -> entry
state stack now 0 3
Entering state 10
Reducing via rule 2 (line 128), definition entry  -> definition
state stack now 0
Entering state 3
Reading a token: --accepting rule at line 159 ("
")
--accepting rule at line 81 ("debug")
Next token is 268 (TOK_DEBUG)
Shifting token 268 (TOK_DEBUG), Entering state 1
Reducing via rule 6 (line 136), TOK_DEBUG  -> debugcmd
```

We'll try to provide better debugging tools in future releases.

7 CFGMON Directive

We're calling a file containing a set of equations and conditions a configuration file; the collection of equations and conditions itself, a configuration. Configuration file names must end in the suffix '`.cfg`'. The name of a configuration is the name of it's file, without the '`.cfg`' suffix.

The simple STOL directive `cfgmon` is used to start and stop processing of configurations. Enter `cfgmon myconfig` to start the configuration monitor running on the file '`myconfig.cfg`'.

The global mnemonic `gbl_cfgmonpath` should be set to a colon-separated list of directories in which the configuration monitor should search to find a configuration file. This is just like the `gbl_procpath` or `gbl_pagepath` mnemonics, only it's for configuration files.

To stop processing a configuration called `myconfig`, enter `cfgmon clear myconfig`.

The special page called `control` can be used to see what configurations are active.

8 Examples

We want to monitor the health of a subsystem and calculate some derived telemetry values for it, but only when the subsystem is turned on. So we run a configuration containing the line:

```
if (p@mysubsysrelay .eq. "on") then cfgmon mysubsys
```

The file 'mysubsys.cfg' then might contain:

```
mypower = (mysubsyscurr * mysubsysvolt)
mytemp = ((mysubsystemp1 + mysubsystemp2 + mysubsystemp3) / 3)
mysomething = (log(sin(mysubsysan1) * cos(mysubsysan2)))
if (p@mysubsysswitch1 .eq. "off" .and. p@mysubsysswitch2 .eq. "off")
        then mystate = 0
if (p@mysubsysswitch1 .eq. "on" .and. p@mysubsysswitch2 .eq. "off")
        then mystate = 1
if (p@mysubsysswitch1 .eq. "off" .and. p@mysubsysswitch2 .eq. "on")
        then mystate = 2
if (p@mysubsysswitch1 .eq. "on" .and. p@mysubsysswitch2 .eq. "on")
        then mystate = 3
if (p@mysubsysrelay .eq. "off") then cfgmon clear mysubsys
```

9 Notes and Bugs

- When equations are *not* being updated on a timed interval, the mnemonic on the left-hand side must not appear in the expression on the right-hand side. Notice that this may be difficult to see:

  ```
  mymnemonic = yourmnemonic * 2
  yourmnemonic = sin(theirmnemonic) / 2
  theirmnemonic = sin(mymnemonic) + cos(hismnemonic)
  ```

  Here, `mymnemonic` depends on `yourmnemonic` which depends on `theirmnemonic` which depends on `mymnemonic`. It's a loop! Since the expressions are evaluated when the mnemonic values are updated, this sort of loop results in rapid-as-possible re-evaluation of the expressions, and a nasty load on the system.

  In a future release, we will try to find these loops and disallow them, but for now, we have to rely on you to do it for us.
- The configuration monitor is not very helpful when it comes to reporting errors in the configuration file. This will improve in future releases.

Index

(Index is nonexistent)

Table of Contents